NAVAL POSTGRADUATE SCHOOL
Monterey, California

THE CONSTRUCTION AND FITTING OF SOME SIMPLE PROBABILISTIC COMPUTER MODELS

by

Donald P. Gaver

January 1975

Approved for public release; distribution unlimited.

ared for:

nse Communications Agency
on, Virginia 22070

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral Isham Linder
Superintendent

Jack R. Borsting
Provost

The work reported herein was supported in part by the Defense Communications Agency, Joint Technical Support Activity and Department of H.E.W. through the Federal Simulation Agency (FEDSIM).

Prepared by:

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NPS55Gv75011	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE *(and Subtitle)*: The Construction and Fitting of Some Simple Probabilistic Computer Models	5. TYPE OF REPORT & PERIOD COVERED: Technical Report
6. PERFORMING ORG. REPORT NUMBER	
7. AUTHOR(s): Donald P. Gaver	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93940	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Defense Communcations Agency, Reston, Virginia 22070	12. REPORT DATE: January 1975
	13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*	15. SECURITY CLASS. *(of this report)*: Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Computer systems
Queueing models
Statistical estimation

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This report describes mathematical models for use in evaluating the performance of complex computer systems. A description is given of the comparison of measurements made on an actual computer, and predictions made from the model.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

THE CONSTRUCTION AND FITTING OF SOME SIMPLE PROBABILISTIC COMPUTER MODELS

Donald P. Gaver
Naval Postgraduate School

1. Introduction.

The purpose of this report is to develop several analytical (probabilistic) computer system models in a form that allows them to be "fitted" to actual measurement data. Some of these models have been validated by comparison with runs of actual sample programs; in general, the agreement achieved has been reasonably good, and suggestions are made here of ways in which the models may be improved so that agreement is likely to be better. Further measurements are being made, and the success with which they can be described by models will be reported later.

The measurement data referred to has been taken on a Honeywell 6000-series computer at the Joint Technical Support Activity (JTSA), a field activity of the Defense Communications Agency in Reston, Virginia. The measurements were made by Mr. Barry Wallack of the JTSA, to whom I am grateful for enthusiastic and careful cooperation.

2. Basic Assumptions.

The mathematical assumption that pervades our analysis is that our systems act as if they are Markovian. That is, at any instant of time, t, we can describe the system state by a random vector $(N_1(t),N_2(t),\ldots,N_k(t)) = \underset{\sim}{N}(t)$; furthermore, knowledge of the value of $\underset{\sim}{N}(t)$ allows us to calculate all future event probabilities. Concretely, $N_1(t)$ may denote the number of programs awaiting and undergoing CPU processing at t, while $N_2(t),N_3(t),\ldots$ represent the number at each of several different memory devices (e.g. disk units or channels thereto, tape units, etc.). The Markovian assumption made is equivalent to assuming that independently and exponentially distributed "burst times" on CPU and memory devices occur. This assumption cannot be rigorously true, but derived results are often insensitive to this assumption (see formulas for the number present in steady state in the infinite server queue; reference [8]). Furthermore, the usual measurements of computer system performance only allow identification of a single rate parameter for a device. Experiments with realistic workload material will be used to check the validity of this assumption as it reflects upon system performance characteristics of interest, such as device utilization, and system throughput.

The reason for making the Markovian assumption is, in the final analysis, to obtain simple, easily understood expressions that involve a few basic parameters. Sensitivity of system performance to departures from this type of assumption can sometimes be effectively assessed by use of another approach, that of diffusion approximation analysis.

3. Simple Configurations.

Actual computer systems, such as the Honeywell 6000 and the IBM 360, must be represented as complex arrays of serving points and queues. Before proceeding to complex models, however, I will describe some much simpler ones. The implications of these can be easily compared to the results of running simple but realistic program material through an actual system. Such validation attempts are currently in progress at JTSA, conducted by Mr. B. Wallack.

Configuration 1: One Processor, One-Channel to Disc.

Suppose one processor (CPU) services J programs that occupy core Assume that the reason that a continuous period of CPU activity on a particular program (a "CPU burst" for short) terminates is the requirement for information contained on a disc. The latter is accessed through a single channel. Because of the presence of other programs in the system, there may well be queueing for the channel. After this queueing is completed, the program accesses the disc, reads out the information, and, at the completion of the IO activity, assumes a place in the CPU queue. The process continues until all computation is completed and the program exits, to be instantly replaced by another. Note that disc mount requests may, in the IBM system, delay the acceptance of new jobs into core. This effect will be modeled later; its effect is not present in the usual simple cyclic models for multiprogramming systems.

Several simplifying assumptions are notable:

(a) there are a fixed number, J, of programs in core,

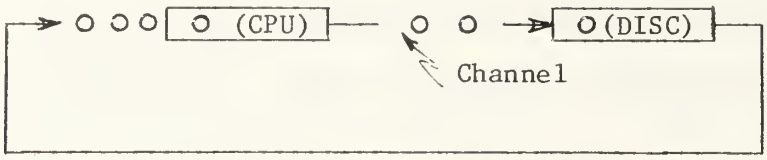
(b) computation and disc activity do not overlap, i.e. I do not here admit the possibility (which apparently exists) that CPU and disc activity can simultaneously occur on the same program,

(c) operating system requirements on the CPU (overhead) are not explicitly included,

(d) Programs are homogeneous enough to be described statistically in a simple fashion.

We allow for these assumptions when the model is compared to actual computer runs of simple program material.

Our configuration appears as below:



Configuration 1.

Analysis. A Markov model of this system requires the specification of the rate parameters of CPU and Channel-Disc: denote the expected length of a CPU burst by λ^{-1}, and of a Disk burst (actual read time from Disc) by μ^{-1}. Furthermore, the system state can be taken to be $N(t)$, the number of programs at the CPU state--waiting plus being served. Obviously $J-N(t)$ are then present at the Disc stage. It may then be shown (see [3], Gaver and Thompson) that (i) there is a long-run or steady-state probability distribution for $N(t)$:

$$p_n = \lim_{t\to\infty} P\{N(t)=n\}$$

where $P\{\quad\}$ denotes a probability measure. Furthermore

$$p_n = p_0\left(\frac{\mu}{\lambda}\right)^n ; \qquad 0 \le n \le J \tag{3.1}$$

$$p_0 = \frac{1 - \frac{\mu}{\lambda}}{1 - \left(\frac{\mu}{\lambda}\right)^{J+1}} \tag{3.2}$$

Discussion. Several points can be made.

(a) The value of p_n is the long-run fraction of time there are exactly n ($n=0$, or 1, or 2, ... or J) programs at the CPU stage. If $\lambda > \mu$ then the jobs tend to be IO bound. If core size is increased, meaning J is increased, then CPU idleness decreases to a positive lower bound:

$$J = 1, \qquad \text{CPU idleness} = p_0 = \frac{\lambda}{\lambda+\mu}$$

$$J = \infty, \qquad \text{CPU idleness} = p_0 = 1 - \frac{\mu}{\lambda} > 0$$

If $\lambda < \mu$, meaning the jobs tend to be CPU bound, then CPU idleness decreases, but now to zero:

$$J = 1, \qquad \text{CPU idleness} = p_0 = \frac{\lambda}{\lambda+\mu}$$

$$J = \infty, \qquad \text{CPU idleness} = p_0 = 0.$$

(b) The system productivity may be assessed as follows. Let program i require b_i CPU bursts, each of expected length λ^{-1}. Then the expected time to finish k programs is

$$\frac{b_1}{\lambda} + \frac{b_2}{\lambda} + \ldots + \frac{b_k}{\lambda} = \frac{1}{\lambda} \sum_{i=1}^{k} b_i \tag{3.3}$$

The expected amount of CPU time available in a time period T is (approximately) equal to $T(1-p_0)$. Thus T must be such that

$$T(1-p_0) = \frac{1}{\lambda} \sum_{i=1}^{k} b_i \tag{3.4}$$

or

$$T = \frac{1}{\lambda(1-p_0)} \sum_{i=1}^{k} b_i \tag{3.5}$$

If the number of bursts per program is thought of as random with mean $\bar{b}$ then

$$T = k\left(\frac{\bar{b}}{\lambda}\right)\frac{1}{(1-p_0)}$$

= (Number of Programs to be Processed) × (Mean CPU Time per Program) ÷ (CPU utilization).

The expression (3.6) is increasingly accurate as k becomes large and as program durations become more homogeneous.

(c) Since p_0 depends only upon the ratio $\frac{\mu}{\lambda}$, and since the number of CPU bursts nearly equals the number of CPU bursts for each program

$$\frac{\text{Mean CPU Time per Program}}{\text{Mean Disc Time per Program}} = \frac{\bar{b}/\lambda}{\bar{b}/\mu} = \frac{\mu}{\lambda} \tag{3.7}$$

The left-hand side of (3.6) can be estimated by running a set of representative programs through an actual system and utilizing a monitor. This has been carried out at JTSA on the Honeywell 6000. Since actual CPU activity includes overhead, and the latter is not explicitly modeled (see Lewis and Shedler, [6], for an explicit model in the IBM context), the rate λ must be reduced to represent overhead.

(d) The expected number of programs at the CPU stage is, in the long run, given by the formula

$$E[N] = \lim_{t\to\infty} E[N(t)] = p_0 \frac{\mu}{\lambda}\left\{\frac{1-\left(\frac{\mu}{\lambda}\right)^J\left[1+J\left(1-\frac{\mu}{\lambda}\right)\right]}{\left(1-\frac{\mu}{\lambda}\right)^2}\right\}.$$

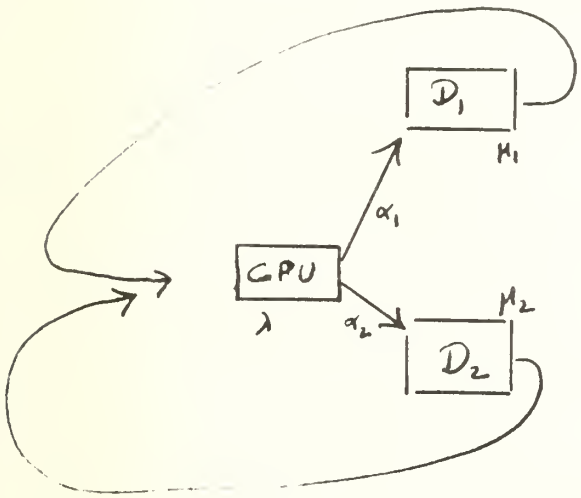
Of course

$$E[\text{number at Channel}] = J - E[\text{number at CPU}]$$

$$= J - E[N].$$

Configuration 2: One Processor, Two Channels to Disjount Groups of Discs.

Again we let one processor serve J programs in a multiprocessing fashion. However, after a program burst terminates we assume that the program requires information from one of a group of Discs, each group being accessed through its own dedicated (single) channel. For the moment we discuss only the two-channel case. The configuration is as shown below.



Under the assumptions (i) that each program that leaves the CPU stage (burst terminates) reports at random--with appropriate probabilities, α_1 and α_2, not necessarily equal to α_1--to Channel 1 and Channel 2, and (ii) that all burst time distributions are exponential, we can analyze the system by means of the Gordon-Newell cyclic queueing model, [5]. It turns out that several relevant measures of system performance can be explicitly written down in parametric form for the present situation; one does not need the more comprehensive methods of Buzen [1].

(a) The Long-Run CPU Idleness Probability.

$$P\{\text{CPU idle}\} = (1-\rho_1)(1-\rho_2)\left[\frac{\rho_2^{J+1} - \rho_1^{J+1}}{\rho_2-\rho_1+(1-\rho_2)\rho_1^{J+2}-(1-\rho_1)\rho_2^{J+2}}\right] \qquad (3.8)$$

where

$$\rho_i = \frac{\lambda\alpha_i}{\mu_i} = \frac{\text{[Channel i (disc) Burst Time]} \times \text{[Probability Channel i Selected]}}{\text{CPU Burst Time}}$$

$i = 1,2$, <u>provided</u> $\rho_1 \neq \rho_2$.

If $\rho_1 = \rho_2$, then

$$P\{\text{CPU idle} \mid \rho_1=\rho_2=\rho\} = \frac{(1-\rho)^2(J+1)\rho^J}{1 - \rho^{J+1}[1+(J+1)(1-\rho)]} \tag{3.9}$$

(b) Long-Run Channel 1 Idleness Probability.

$$P\{\text{Channel 1 idle}\} = (1-\rho_1)(1-\rho_2^{J+1})\left[\frac{\rho_2 - \rho_1}{\rho_2-\rho_1+(1-\rho_2)\rho_1^{J+2}-(1-\rho_1)\rho_2^{J+2}}\right] \tag{3.10}$$

A corresponding expression for Channel 2 is obtained by simply interchanging ρ_1 and ρ_2 in the above formula.

(c) Long-Run Expected Number at Channel, and at CPU, Stages.

It can be shown that the expected number at the channel stage, both enqueued and in process of device access, is given by the expression

$$E\{\text{Number at channels}\} = \frac{(1-\rho_2)(1-\rho_1)}{\rho_2-\rho_1+(1-\rho_2)\rho_1^{J+2}-(1-\rho_1)\rho_2^{J+2}}\left[\rho_2^2\left\{\frac{1-(J+1)\rho_2^J+J\rho_2^{J+1}}{(1-\rho_2)^2}\right\} - \rho_1^2\left\{\frac{1-(J+1)\rho_1^J+J\rho_1^{J+1}}{(1-\rho_1)^2}\right\}\right] \tag{3.11}$$

Of course

$$E[\text{number at CPU}] = J - E[\text{number at Channels}].$$

Note that the above formulas do not give the individual occupancies (queues plus those in service) at the separate channels, but only the sum of those occupancies.

(d) Long-Run Expected Number at Channel 1.

It can be shown that the expected number of programs at Channel 1 is given by the expression

$$E[\text{number at Channel 1}] =$$

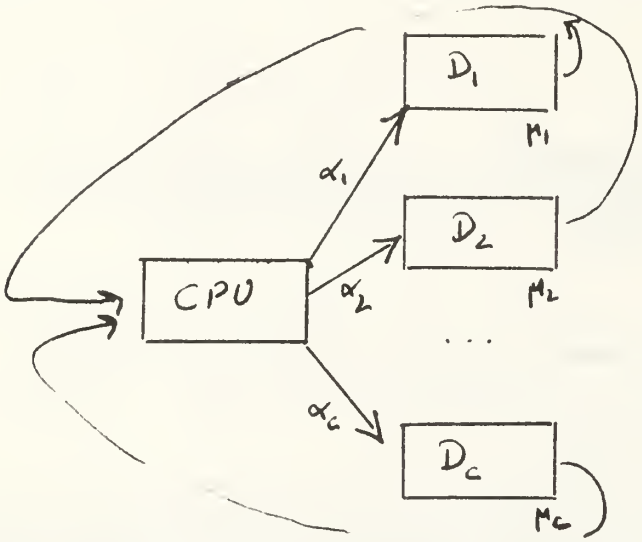
$$\frac{(1-\rho_1)(\rho_2-\rho_1)}{\rho_2-\rho_1+(1-\rho_2)\rho_1^{J+2}-(1-\rho_1)\rho_2^{J+2}}\left[\rho_1\left\{\frac{1-(J+1)\rho_1^{J}+J\rho_1^{J+1}}{(1-\rho_1)^2}\right\}-\rho_1\rho_2^{J}\left\{\frac{1-(J+1)(\rho_1/\rho_2)^{J}+J(\rho_1/\rho_2)^{J+1}}{(1-\rho_1/\rho_2)^2}\right\}\right] \quad (3.$$

The expected number present at Channel 2 can be obtained by reversing ρ_1 and ρ_2 in this formula.

Note that this expression and the one given previously may not describe the (expected) number present if the latter number is obtained from observations made at special instants of time. For instance, if the Channel occupancy is counted just <u>before</u> each new arrival to its queue occurs one would anticipate an average lower than that resulting from counts made just <u>after</u> each new arrival: think of the case $J = 1$, for instance. Consequently the formulas in (c) and (d) may require adjustment in order account for sampling techniques actually used. As they stand, they describe the expected channel occupancy when the channel is observed either continuously through time, or at uniformly random time points.

Configuration 3: One Processor, c Channels to Disjoint Groups of Discs.

The figure below depicts a very common multiprogramming computer configuration at least in an approximate manner.



A program or job experiences a burst of mean duration λ^{-1} at the CPU and then selects a Channel to a group of devices; Channel i is selected with probability α_i, independently of all previous events. Thus each Channel has its own queue, and the CPU also has a queue. The service rate on Channel i is μ_i. There are assumed to be J jobs simultaneously in the system, and we study the process when it is in the equilibrium state.

Analysis: The Gordon-Newell Cyclic Queue Model.

In [5] Gordon and Newell present a general solution to the multi-dimensional Markov process describing the above setup. Letting N_i be a random variable denoting the number present at Channel i in equilibrium we can show that for the present configuration

$$\begin{aligned} p(n_1,n_2,\ldots,n_c) &= P\{N_1=n_1,N_2=n_2,\ldots,N_c=n_c\} \\ &= K(J)\rho_1^{n_1}\rho_2^{n_2}\ldots\rho_c^{n_c}, \qquad n_1+n_2+\ldots+n_c \le J. \end{aligned} \tag{3.13}$$

where K(J) is determined by normalization, i.e. by the condition that

$$\sum_{n_1+n_2+\ldots+n_c \le J} \cdots \sum p(n_1,n_2,\ldots,n_c) = 1. \tag{3.14}$$

We shall derive an easily computed explicit formula for $K(J)$ in the present case. Let

$$\rho_i = \frac{\lambda\alpha_i}{\mu_i}, \qquad i = 1,2,\ldots,c \tag{3.15}$$

the traffic intensity parameter for the single-server queue at Channel i. Recall that if Channel i were confronted by a Poisson arrival rate of $\lambda\alpha_i$ then the equilibrium distribution is geometric (if $\rho_i < 1$), and indeed if J becomes very large, so that the CPU is constantly busy, then it is just such an equilibrium distribution that is seen. In fact, we can write

$$P\{\tilde{N}_1=n_1,\tilde{N}_2=n_2,\ldots,\tilde{N}_c=n_c \mid \tilde{N}_1+\tilde{N}_2+\ldots+\tilde{N}_c \le J\} = \frac{(1-\rho_1)\rho_1^{n_1}(1-\rho_2)\rho_2^{n_2}\ldots(1-\rho_c)\rho_c^{n_c}}{P\{\tilde{N}_1+\tilde{N}_2+\ldots+\tilde{N}_c \le J\}} \tag{3.16}$$

where $\tilde{N}_i$ is the state variable of an unlimited Poisson arrival queue, and note that

$$p(n_1,n_2,\ldots,n_c) = \frac{(1-\rho_1)(1-\rho_2)\ldots(1-\rho_c)}{P\{\tilde{N}_1+\tilde{N}_2+\ldots+\tilde{N}_c \le J\}} \rho_1^{n_1}\rho_2^{n_2}\rho_c^{n_c}. \tag{3.17}$$

Thus all that is necessary to find the normalizing constant is to find the distribution of the sum $\tilde{N}_1 + \tilde{N}_2 + \ldots \tilde{N}_c = \tilde{N}$. This step is facilitated by taking generating functions:

$$E[z^{\tilde{N}_i}] = (1-\rho_i) \sum_{n_i=0}^{\infty} z^{n_i}\rho_i^{n_i} = \frac{1-\rho_i}{1-z\rho_i}, \tag{3.18}$$

and thence, by the convolution theorem

$$E[z^{\tilde{N}}] = \prod_{i=1}^{c} \frac{1-\rho_i}{1-\rho_i z_i} = g(z). \tag{3.19}$$

Now suppose all ρ_i are numerically different, as will often be true. It is convenient to carry out a partial fractions development as follows: rewrite $g(z)$ as

$$g(z) = \sum_{i=1}^{c} \frac{a_i}{1-\rho_i z} \tag{3.20}$$

and then determine a_i as follows: multiply $g(z)$ by $1 - \rho_i z$ and let $z \to \rho_i^{-1}$. From the <u>product</u> form for $g(z)$ we obtain after cancelling the term going to zero,

$$\frac{\prod_{i=1}^{c} (1-\rho_i)}{\prod_{j=1}^{c}{}^{(i)} (1-\rho_j/\rho_i)} = a_i \tag{3.21}$$

where the right-hand side comes from cancelling off the denominator $1 - \rho_i z$ term against the numerator of $(1-\rho_i z)g(z)$; $\prod_{i=1}^{c}{}^{(i)}$ means the $i^{\underline{th}}$ factor is omitted from the product. Thus we finally have $g(z)$ completely determined, and a term-by-term geometric expansion gives the probability distribution of $\tilde{N}$:

$$P\{\tilde{N}=n\} = \sum_{i=1}^{c} a_i \rho_i^n . \tag{3.22}$$

Next

$$P\{\tilde{N}\leq n\} = \sum_{i=1}^{c} a_i \sum_{n'=0}^{n} \rho_i^{n'} = \sum_{i=1}^{c} a_i \frac{[1-(\rho_i)^{n+1}]}{1-\rho_i} \tag{3.23}$$

and we can put $n = J$ to obtain the denominator of the expression for $p(n_1,\ldots,n_c)$, and hence the normalizing constant. The explicit expression is thus

$$p(n_1,n_2,\ldots,n_c) = \left[\sum_{i=1}^{c}\left\{\frac{1-\rho_i^{J+1}}{1-\rho_i}\right\}\prod_{j=1}^{c}{}^{(i)}\ \frac{\rho_i}{\rho_i-\rho_j}\right]^{-1} \rho_1^{n_1}\rho_2^{n_2}\ldots\rho_c^{n_c} \qquad (3.24)$$

$$n_1 + n_2 + \ldots + n_c \le J.$$

Several formulas for useful operational measures can easily be derived, e.g. from the observation that the d.f. of $N = N_1 + \ldots + N_c$ is the same as that of $\tilde{N}_1 + \ldots + \tilde{N}_c = \tilde{N}$, given $\tilde{N} \le J$. For instance, the probability that the CPU is idle is the probability that $N = J$, so all jobs are at the Channel stage; now in order to obtain a formula, write

$$P\{\tilde{N}=J\} = \sum_{i=1}^{c} a_i\rho_i^J \qquad (3.25)$$

and then

$$P\{N=J\} = P\{\tilde{N}=J \mid \tilde{N}\le J\} = \frac{\sum_{i=1}^{c} a_i\rho_i^J}{\sum_{i=1}^{c} a_i\,\frac{1-\rho_i^{J+1}}{1-\rho_i}} = \frac{\sum_{i=1}^{c} a_i\rho_i^J}{1-\sum_{i=1}^{c} a_i\,\frac{\rho_i^{J+1}}{1-\rho_i}}. \qquad (3.26)$$

In summary

(a)

$$p_0(c,J) = \frac{\sum_{i=1}^{c} a_i\rho_i^J}{1-\sum_{i=1}^{c} a_i\,\frac{\rho_i^{J+1}}{1-\rho_i}} \qquad (3.27)$$

is the long-run fraction of time the CPU is idle in a c-Channel system with J jobs in core. Here

$$a_i = \left[\prod_{i=1}^{c} (1-\rho_i)\right]\left[\prod_{j=1}^{c}{}^{(i)} \left(\frac{\rho_i}{\rho_i - \rho_j}\right)\right] \tag{3.28}$$

where $\Pi^{(i)}$ represents the product with the term $j = i$ omitted.

(b) The total Turnaround Time for servicing k programs equals the Total CPU Time, divided by $1 - p_0(c,J)$.

(c) The system under study is closed (actually, completed programs are assumed to leave and then to be instantly replaced), and so a long-run steady state is reached. For such a Markovian system we know that a condition of detailed balance exists:

$$P\{\text{CPU Busy}\}\lambda p_i = P\{\text{Channel } i \text{ Busy}\}\mu_i. \tag{3.29}$$

To explain this intuitively, suppose we run the system for a long time, t. Then the CPU is busy (not idle) for a time approximately equal to

$$t \cdot P\{\text{CPU Busy}\} = t[1-p_0(c,J)]; \tag{3.30}$$

the latter is a consequence of renewal theory arguments; see Cox [2]. Now at any moment during a CPU busy time a departure occurs with approximate probability λdt, and that departure represents a program destined for Channel i with probability α_i. Consequently the long-run expected number of programs that flow towards Channel i is

$$t\ P\{\text{CPU Busy}\}\lambda\alpha_i$$

But the long-run expected number of programs that flow from Channel i to the CPU is $t\ P\{\text{Channel } i \text{ Busy}\}\mu_i$, for all programs that leave Channel i go back to the CPU. And we must have balance:

Number of Programs from CPU to Channel i =

Number of Programs from Channel i to CPU

in the long run, else a buildup will occur at one point or the other. Thus

$$\underline{t}\ P\{\text{CPU Busy}\}\lambda\alpha_i = \underline{t}\ P\{\text{Channel i Busy}\}\mu_i$$

or

$$\frac{\underline{t}\ P\{\text{Channel i Busy}\}}{\underline{t}\ P\{\text{CPU Busy}\}} = \frac{\lambda\alpha_i}{\mu_i} = \rho_i \tag{3.31}$$

or, approximately,

$$\frac{\text{Long-run Busy Time for Channel i}}{\text{Long-run Busy Time for CPU}} \approx \frac{\lambda\alpha_i}{\mu_i} = \rho_i \tag{3.31,a}$$

This approximate equality allows us to estimate ρ_i, the input to our evaluation equation, from monitor data that records total CPU and Channel times. We shall put it to use shortly.

(d) The expected number of programs at the Channel stage can be calculated as follows (note that N here denotes the total of all jobs in residence at the Channel stage in the long run; so J - N is the number at the CPU):

$$= \frac{\sum_{n=0}^{N} nP\{N=n\}}{\dfrac{\sum_{n=0}^{J} n \sum_{i=1}^{c} a_i\rho_i^n}{1 - \sum_{i=1}^{c} a_i \dfrac{\rho_i^{J+1}}{1-\rho_i}}}$$

$$= \frac{\sum_{i=1}^{c} a_i\rho_i \dfrac{1-\rho_i^J(1+J-\rho_i J)}{(1-\rho_i)^2}}{1 - \sum_{i=1}^{c} a_i \dfrac{\rho_i^{J+1}}{1-\rho_i}}$$

A similar formula is available for the expected number present at each of the i Channels.

The above formulas may easily be evaluated numerically. Computer programs for so doing have been written in FORTRAN and run on the IBM 360/67 system at the Naval Postgraduate School. In addition, Professor Neagle Forrest has programmed the two-channel formulas for the HP-65 hand-held calculator. Extension of this latter program to more complex and realistic configurations would allow portable use of the models by sizing teams.

4. Fitting the Models to Measurement Data.

Next I embark upon a discussion of the manner in which some actual program material, constructed during the fall of 1974 by Mr. B. Wallack of JTSA, behaved (a) when submitted to the Honeywell 6000, being allowed 128K of core, and (b) was estimated to behave by the simple analytic (Markov) models that have been developed. This discussion is concerned with the first of our attempts to validate, or at least test the consistency and perhaps the usefulness of, our simple models.

A. The Program Material.

Program material submitted to the H6000 consisted of 40 programs that required CPU processing alternating with disc file accesses. Each program was so constructed that all CPU "burst" lengths were identical, but burst lengths varied between programs. The set of programs was submitted as a batch to the H6000 and the following items were recorded.

(1) Total Job (Problem Program) CPU Time.

(2) Total System CPU Time.

(3) Total Disc Time (Single Channel Case).

(4) Total Channel Time for Each Channel (Two Channels to Two Discs Case).

(5) Average CPU Queue (including item being processed).

(6) Average Channel A Queue (including item being processed; apparently the queue was examined just prior to each moment a job joined the channel queue).

(7) Average Channel B Queue (same as (6) above).

(8) Total Turnaround Time (to finish all 40 jobs).

(9) Total CPU Idle Time.

(10) Average Number of Programs in Core (later, a "core map" describing the number of programs in core at 30 second intervals became available).

B. Fitting the Models.

It is possible to fit the models to the data furnished by making use of the measured data described. I will describe such fits, and comment upon them.

First note that formulas for CPU idleness probability use as basic parameter the ratio

$$\frac{\lambda}{\mu} = \frac{\frac{1}{\mu}}{\frac{1}{\lambda}} = \frac{\text{Expected Disc or I/O burst time}}{\text{Expected CPU burst time}}.$$

Under the assumption that CPU and I/O bursts essentially alternate (certainly not always accurate) there are very nearly as many CPU bursts as I/O bursts, so

$$\frac{\lambda}{\mu} \approx \frac{\text{Total Disc Time}}{\text{Total CPU Time}} ;$$

for present purposes we have added items (1) and (2) to obtain Total CPU Time, effectively lowering the CPU processing rate to account for overhead. A more sophisticated (but still simple) model has been constructed for overhead; we do not attempt to fit it here.

Next observe that our formulas for CPU idleness pretend that a fixed number J, of programs is in core. This is literally untrue, but we shall set the average of the measured number in core equal to J, later examining the effect of this step (better possibilities suggest themselves).

Finally, we will calculate CPU utilization via our formulas, and utilize the latter to estimate total turnaround time for the 40-program job stream. Other comparisons will also be made.

C. Actual Cases.

Case 1: One CPU, One Channel, One Device

Job CPU Time = 0.9830

System CPU Time = 0.0348

Disc Time = 0.8407

Average Number of Jobs in Core = 3.1

It follows that we estimate

$$\frac{\lambda}{\mu} = \frac{0.8507}{0.9830 + 0.0348} = 0.836$$

We may use the basic formula (3.2) and approximate relationship (3.31,a) to estimate CPU idleness probability. The value obtained is

$$p_0 = 0.181$$

Total (expected) turnaround time is then estimated by dividing total CPU time by CPU utilization:

$$\text{Estimated Expected Turnaround Time} = \frac{0.9830 + 0.0348}{1 - 0.181}$$

$$= \frac{1.0178}{0.819} = 1.24.$$

Total measured turnaround time reported was 1.41; the predicted result was about 12% below the actual. This quality of agreement is encouraging, considering the discrepancies between the model and the actual program material. It is worthwhile to search for explanation of the discrepancy, however.

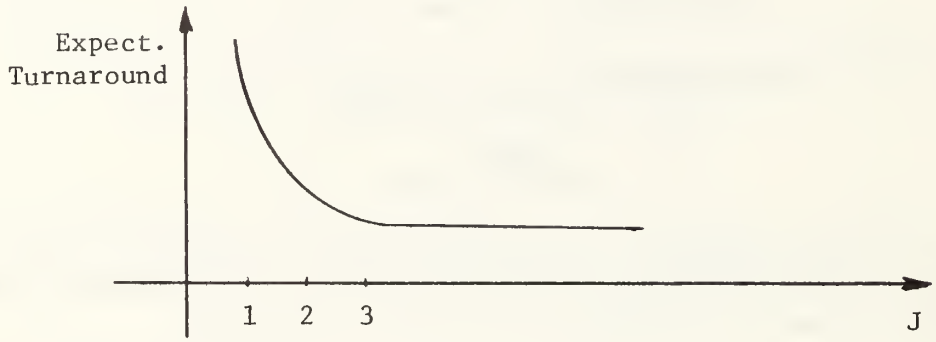
Sensitivity to Core Occupancy, J. It is easily possible to compute the effect of various core occupancies; we do so for only two additional values:

$$d_2 = 0.275, \qquad \text{and} \qquad d_4 = 0.136$$

$$\text{Expected turnaround time, } J = 2 = \frac{1.0178}{0.725} = 1.40,$$

$$\text{Expected turnaround time, } J = 4 = \frac{1.0178}{0.864} = 1.18.$$

If we graph expected turnaround time as a function of J the following type of curve results:



Consequently if we have the probability distribution of J, and average first, and use the latter average in d_J to estimate turnaround time we obtain a result that is smaller than that obtained by averaging turnaround time, conditional on J. Thus, it may be that careful attention to the core occupancy stochastic process will deliver more accurate predictions of turn around time.

In order to study this effect further a "core map" at 30 second intervals was made by B. Wallack. This allowed empirical development of the distribution of core content for the data and configuration under study here. We find (letting $p(J)$ denote the long-run probability that J programs are in core simultaneously) the following numerical values.

Number of Programs in Core (J)	Frequency (estimated p(J))
1	0.085
2	0.235
3	0.268
4	0.386
5	0.026

Under the apparently valid supposition that number of programs in core (core occupancy) changes slowly as compared to "service," i.e. CPU and I/O activities, we can argue that for fraction of time $p(J)$ the output rate is essentially $\lambda[1-d_J]$, so expected turnaround or throughput time requires the average or expectation

$$E[T_k] \approx \frac{k\bar{b}}{\lambda} \sum_{J=1}^{\infty} \frac{1}{1 - d_J} \; p_J$$

where k = number of programs, and $\bar{b}$ = average CPU time per program. The JTSA experiment essentially estimates $\frac{k\bar{b}}{\lambda}$ = total CPU time. Now for the particular data in question we then computed $(1-d_J)^{-1}$ for $J = 1,\ldots,5$:

J	$(1-d_J)^{-1}$
1	1.8360
2	1.3807
3	1.2305
4	1.1566
5	1.1132

Weighting the above by the estimated p_J's, and finally multiplying by total CPU time yields an estimated turnaround time of 1.31, which is within 7% of what was actually observed, although still low.

Comments.

Variation in core occupancy may account for the observed discrepancy between simple formula (constant occupancy) turnaround estimates and actual measurements. Required are (a) methods for estimating the error of our turnaround estimate, since the latter is based on experience with finitely many programs, (b) a model for predicting the core occupancy distribution, $\{p(J)\}$, when core size is changed, (c) checks of results when CPU burst times and Disc times are not exponentially distributed.

Case 2: One CPU, Two Channels to Two Devices, Programs Utilize One Device Exclusively.

Programs were so written as to make use of just one device; this demand pattern seems to make the model's Markovian-random demand structure less applicable. Results of model fitting (assuming constant J) are as follows.

Measured Total CPU Time = 1.3044

Measured Total Channel A Time = 0.4089

Measured Total Channel B Time = 0.4425

$\hat{\rho}_1 = 0.3135$, $\hat{\rho}_2 = 0.3392$, $\hat{J} = 2.9$

(We use $\hat{}$ to denote an estimate from data available.) If these estimates are used to obtain

	Measured	Model Output
p_0	0.072	0.0136
Turnaround	1.32	1.41
CPU Queue	1.97	2.19

Although the model-predicted CPU idleness differs from that measured by a factor of five, the turnaround times are reasonably close.

Case 3: One CPU, Two Channels to Three Devices; Programs Utilize One Device Exclusively.

The system configuration in this case allows programs to access two disc devices by means of one channel, and another disc by means of a second channel. Core size was apparently increased in this exercise also, for the average core occupancy was slightly over six jobs. Results obtained from measurement and model fitting are as follows.

Measured Total CPU Time = 1.1429

Measured Total Channel A (16) Time = 0.6012

Measured Total Channel B (8) Time = 0.2489

$\hat{\rho}_1 = 0.5260$, $\hat{\rho}_2 = 0.3392$, $\hat{J} = 6.16$

If these estimates are used as input we obtain

	Measured	Model Output
p_0(CPU Idleness)	0.0246	0.0122
T (Turnaround)	1.18	1.16
CPU Queue (expected)	4.86	4.71
Device Queue (expected)	1.30	1.29

Although the assumptions of the model are not well satisfied for this particular workload, the model and measurements generally agree well, although predicted idleness probability is lower than that measured by a factor of two.

Further exercises are being conducted and analyzed, and the results will be reported as they become available. In general, there has been reasonable agreement between model-predicted results and the measurements made on Honeywell equipment that have been furnished to me.

5. Statistical Considerations in Fitting Computer Models to Data.

The previous section described the quality of the fit obtained when the simple Markovian computer model was fitted to actual (Honeywell) workload data. The purpose of this section is to consider some of the problems of statistical estimation that arise when considering model fitting and assessment.

A. Inferences from Measurements: Sources of Error.

A-1. Sampling Error.

In order to obtain measurements of system performance, strings of $k = 40$ programs were run through each actual computer configuration. It is a truism that results obtained from a particular set of 40 programs would not be exactly reproduced if other sets of similar jobs were run.

It is of interest to use the output data actually obtained from a given sample to set rough confidence limits on an unknown expected turnaround time. In order to do this, the following steps were taken

(i) Mr. Wallack of JTSA furnished me with the clock times at which each of the 40 jobs left the system. These were naturally arranged in increasing order; differences between successive outputs were found by subtraction.

(ii) The time differences between successive system departures were examined statistically. In particular frequency plots were made of the distribution of the time between successive departures, and moment estimators were computed.

(iii) Estimates of mean inter-departure time and standard derivation of inter-departure time were in reasonably close numerical agreement (standard deviation slightly smaller than mean), which indicates that program exits

occur at nearly exponentially distributed time intervals. Such is what one anticipate if a "thinning" mechanism is in effect: with small probability θ a program exits the system after a CPU burst (to be replaced by one enqueued) while with probability $1 - \theta$ it approaches a channel, later to return to the CPU. If exit events are independent, i.e. Bernoulli trials, then under our Markov assumptions actual departures should be close to exponentially distributed. Assuming equality of mean and standard deviation, then, we conclude that total turnaround time, T, is approximately gamma distributed, on the basis of which deduction confidence limits are obtainable. In fact, total observed turnaround time estimates $E[T]$, expected turnaround time, and $\sqrt{k}$ × total observed turnaround estimates $[Var[T]]^{1/2}$. Since $k = 40$ is large enough to justify a normal approximation, we state that, with approximately 95% confidence, $E[T]$ lies in the interval $[(\text{observed turnaround})(1 \pm \frac{2}{\sqrt{k}})] = [(\text{observed turnaround})0.68, (\text{observed turnaround})(1.32)]$.

On the basis of the above error assessment it appears that a sample of 40 jobs is sufficient to estimate $E[T]$ with an error of about $\pm 30\%$; quadrupling the batch size will reduce this error to $\pm 15\%$, with 95% confidence.

It would seem to be worthwhile to keep the above figures in mind when models are being assessed--and, for that matter, when systems are being sized simply by running representative experimental programs and judgmentally interpreting the results. Although there are alternatives to the error assessment presented, the latter is simple and requires only the total turnaround time statistic. I plan to study more complex, and possibly more accurate, alternatives in the future.

Turn now to a consideration of other possible causes of real or apparent model error.

A-2. Core Occupancy Tail-off at End of Run.

Observe that if we knew ρ_i, for $i = 1,2,\ldots,c$, and core occupancy, J, remained constant, and further that our models' assumptions (Markovian-exponential) are well satisfied, then the long-run formulas will predict E[T] accurately provided the number of programs, k, is "large" enough so that transients at the beginning and end of a run are insignificant. None of the above assumptions are strictly true, and an examination of core occupancy data ("core maps") shows that there is some tailing off of occupancy as the end of a run is approached. Just how serious this effect is upon our estimates is under investigation at JTSA. Such tail off should tend to artificially prolong experimental runs.

A-3. Core Occupancy: Slow Variations.

The assumption that the number of programs in core is fixed and equal to the expected (average) number of jobs in core is an oversimplification, and tends to produce a constant bias, as has been remarked. A model that addresses this problem is under construction.

A-4. Program Material Statistics.

If CPU and Channel-Device burst times do not resemble independent exponential random variables then systematic prediction errors are likely to occur. In particular, if CPU bursts tend to be hyper-exponential (skewed, long-tailed) appearing, as may be the case when mixtures of programs are present, then the Markov model understates CPU idleness and turnaround time; see Gaver [2], and Gaver and Shedler [4]. Before a more refined model can be fitted, however, further measurements must be made and interpreted.

6. A Markovian Model that Includes Overhead.

The purpose of this section is to derive a simple model for a multiprogramming computer system that explicitly includes some of the possible effects of "overhead" actions by the CPU. It will be assumed that at the termination of every activity "burst," whether it be CPU or I/O, the CPU enters an "overhead" state, residing there for a period long enough to carry out activity so described.

In order to keep the analysis and formulas simple the derivations will be carried out for a Markovian system. Our objective is to see, at least qualitatively, how overhead decreases overall system productivity. The present results, although based on oversimplified assumptions, are perhaps a degree more realistic than is the simple ad hoc procedure of reducing CPU production rate, λ, by an empirically derived factor.

A. Derivation.

Let $N(t)$ denote the number of programs at (waiting, and being served by) the CPU at time t. Let

$$P_j(t) = P\{N(t) = j, \text{ and CPU doing problem program computing}\}.$$

$$Q_j(t) = P\{N(t) = j, \text{ and CPU performing overhead}\}.$$

Let ν^{-1} be the expected duration of an overhead activity (any overhead activity, according to our present simple setup). Furthermore, assume that at the termination of any problem program work, be it at CPU or I/O, the CPU is immediately preempted for a random, exponentially distributed time of mean duration ν^{-1}.

If j programs are at the CPU, let λ_j denote the CPU program burst completion rate, and let μ_j be the corresponding rate for I/O. Later on we can fill in specific functions for λ_j and μ_j; at present it seems unnecessary.

Now write down the basic probability equations for $P_j(t)$ and $Q_j(t)$:

$$P_j(t+dt) = P_j(t)[1-(\lambda_j+\mu_j)dt] + \nu Q_j(t). \tag{6.1}$$

the latter following because if the system is in CPU-active states $j+1$ or $j-1$, and CPU or I/O completes, then the system can only go into CPU-overhead state j, and thence to active state j. Next

$$Q_j(t+dt) = Q_j(t)[1-\nu dt] + P_{j+1}(t)\lambda_{j+1}dt + P_{j-1}(t)\mu_{j-1}dt; \tag{6.2}$$

the above explanation covers the latter equation.

Conversion to differential equations, and thence to balance equations, is routine (set $\frac{dP_j}{dt} = 0$, $\frac{dQ_j}{dt} = 0$ to get balance). The balance equation results are:

$$p_j(\lambda_j+\mu_j) = \nu q_j, \qquad \nu q_j = \lambda_{j+1}p_{j+1} + \mu_{j-1}p_{j-1} \tag{6.3}$$

Therefore, equating νq_j terms gives

$$p_j(\lambda_j+\mu_j) = \lambda_{j+1}p_{j+1} + \mu_{j-1}p_{j-1}. \tag{6.4}$$

Start with

$$\lambda_0 = 0 = \mu_{-1}$$

$$p_0\mu_0 = \lambda_1 p_1 \tag{6.5}$$

and repeatedly solve:

$$p_1 = p_0 \frac{\mu_0}{\lambda_1}$$

$$\cdots$$

$$p_j = p_0 \frac{\mu_0\ \mu_1 \cdots \mu_{j-1}}{\lambda_1\ \lambda_2 \cdots \lambda_j} \tag{6.6}$$

and

$$q_j = p_j\left(\frac{\lambda_j+\mu_j}{\nu}\right) = p_0 \frac{\mu_0\ \mu_1 \cdots \mu_{j-1}}{\lambda_1\ \lambda_2 \cdots \lambda_j}\left(\frac{\lambda_j+\mu_j}{\nu}\right). \tag{6.7}$$

Finally,

$$\sum_{j=0}^{\infty} (p_j+q_j) = 1. \tag{6.8}$$

B. Example.

Suppose J jobs multiprogram and make use of a single channel to various devices. Therefore, $\lambda_j = \lambda$, $\mu_j = \mu$, and

$$p_j = p_0\left(\frac{\mu}{\lambda}\right)^j$$

$$q_j = p_0\left(\frac{\mu}{\lambda}\right)^j\left(\frac{\lambda+\mu}{\nu}\right) \tag{6.9}$$

from (6.6) and (6.7). Summing up from $j = 0$ to J gives, with the aid of (6.8),

$$p_0 = \frac{\nu}{\nu+\lambda+\mu} \quad \frac{1 - \frac{\mu}{\lambda}}{1-\left(\frac{\mu}{\lambda}\right)^{J+1}} \tag{6.10}$$

The long-run fraction of time during which the CPU is concerned with overhead activity is

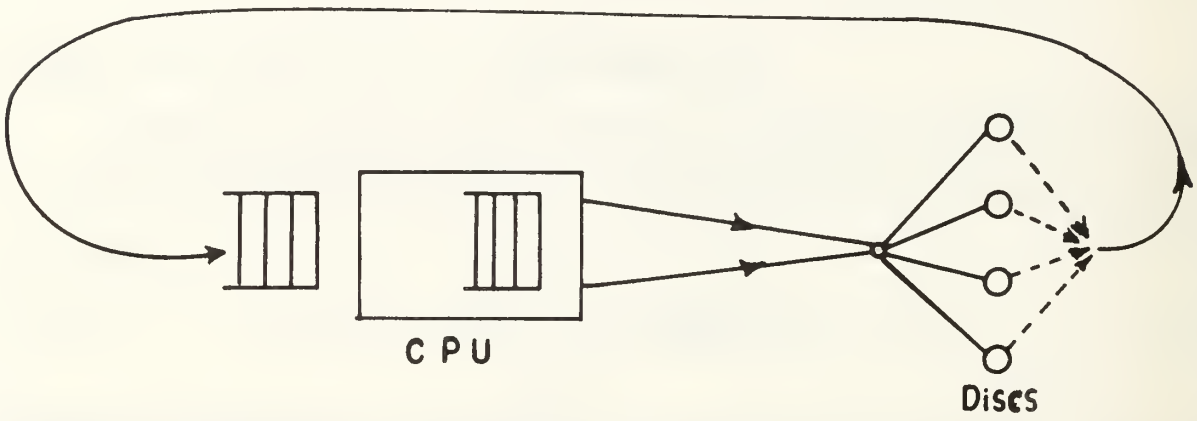
$$\sum_{j=0}^{\infty} q_j = p_0\left(\frac{\lambda+\mu}{\nu}\right) \sum_{j=0}^{J} \left(\frac{\mu}{\lambda}\right)^j$$

$$= \frac{\lambda+\mu}{\lambda+\mu+\nu} . \tag{6.11}$$

7. Conflict on Channels Leading to Memory Devices.

A. The Setup: Two Channels to Several Equi-Demanded Discs

Consider the following configuration, describing a system with a CPU, several memory devices (e.g., discs), and two channels by way of which the devices are accessed.



In conformity with Markov assumptions let λ be the CPU service rate, so λ^{-1} is an expected CPU burst, and let μ^{-1} be the expected device burst.

It will be assumed first here that at the termination of any CPU burst the program is equally likely to proceed to any device. There may well be an attempt to ensure this type of behavior by shifting filed information around to equalize device appeals. Other assumptions (unequal probabilities) are more difficult to handle, and must wait. They can be handled in an analytical framework, but more equations must be solved.

Further assume that, so long as there is no conflict for a particular device's information, the rate of service is--where n

now denotes the number of programs in the channel-device stage--

$$\mu_n = \begin{cases} 0 & \text{if} \quad n = 0 \\ \mu & \text{if} \quad n = 1 \\ 2\mu & \text{if} \quad n \geq 2. \end{cases} \tag{7.1}$$

If, on the other hand, a request comes from the CPU for information from the same disc or one already in use, and reaches the head of the device line, then

$$\mu_n = \begin{cases} 0 & \text{if} \quad n = 0 \\ \mu & \text{if} \quad n \geq 1. \end{cases} \tag{7.2}$$

If there are D devices in all, the chance of a conflict is assumed here to be D^{-1}; this assumption will be changed if necessary.

Assume also that service is in arrival order, so that if a channel is connected to device i in order to service a particular program, and if a program immediately behind the one being served also requires the disc in question, then effectively the second channel is blocked.

β. Probability Model

Let (a) $P_n(t)$ denote the probability that n programs are present at the device stage (waiting and in service) and both channels are available (no blocking), while (b) $B_n(t)$ denotes the probability of the same event, save for the difference that one channel is blocked.

Let λ be the CPU burst rate (λ^{-1} = expected CPU burst time), and μ^{-1} be the expected time for which a disc is continuously engaged by a single program. We let the probability that a program is blocked be β ($\beta = \frac{1}{D}$, D being the number of discs), and $\bar{\beta} = 1-\beta$.

B-1 Probability Equations

(1) $$P_0(t+dt) = P_0(t)[1-\lambda dt] + P_1(t)\mu dt + o(dt), \qquad (7.3)$$

for no blocking is possible when $n = 0$; the first term on the right hand side (RHS) represents the probability that the number of programs at the device-wait stage is 0, and no new arrival occurs in $(t,t+dt)$, while the second is the probability that 1 was present, occupying a channel, but terminated in $(t,t+dt)$.

(2) $$P_1(t+dt) = P_1(t)[1-(\lambda+\mu)dt] + P_0(t)\lambda dt + P_2(t)2\mu dt + B_2(t)\mu dt + o(dt). \qquad (7.4)$$

The only term requiring remark is the last on (RHS): $B_2(t)$ is the probability that 2 are present, but only one channel is in use (blocking), but in $(t,t+dt)$ the one in service leaves, and either channel is now open.

(3) $$P_2(t+dt) = P_2(t)[1-(\lambda+2\mu)dt] + P_1(t)\lambda\bar{\beta}dt + P_3(t)2\mu\bar{\beta}dt + B_3(t)\mu\bar{\beta}dt + o(dt). \qquad (7.5)$$

The first RHS term represents no change from 2. The second term represents the probability of 1 present and a non-blocked arrival. The third term means that 3 are present, 2 of which are in service; a departure occurs, and the $3^{\underline{rd}}$ is non-blocking. The fourth term means 3 present, one channel blocked hence one in service, a departure occurs in $(t,t+dt)$ and the next two do not require the same disc.

In general, for $n > 2$ an arrival doesn't enter service, so

(4) $$P_n(t+dt) = P_n(t)[1-(\lambda+2\mu)dt] + P_{n-1}(t)\lambda dt + P_{n+1}(t)2\mu\bar{\beta}dt + B_{n+1}(t)\mu\bar{\beta}dt + o(dt) \qquad (7.6)$$

Turn now to equations for $B_n(t)$. There can be no blocking for $n = 0$ or 1, so $B_0(t) = B_1(t) = 0$. We omit writing $o(dt)$.

(5) $$B_2(t+dt) = B_2(t)[1-(\lambda+\mu)dt] + P_1(t)\lambda\beta dt + B_3(t)\mu\beta dt + P_3(t)2\mu\beta dt. \qquad (7.7)$$

The first term represents no change (only one is in service). The second represents 1 in service and the arrival of a program requiring the same disc. The third represents 3 present, 1 in service and one channel blocked; the departure of the latter finds that the next in line is blocked. The fourth means 3 present, 2 in service; one departs, and the third is blocked.

For $n \geq 3$,

$$B_n(t+dt) = B_n(t)[1-(\lambda+\mu)dt] + B_{n-1}(t)\lambda dt + B_{n+1}(t)\mu\beta dt + P_{n+1}(t)2\mu\beta dt \qquad (7.8)$$

To derive differential equations, subtract $P_n(t)$ or $B_n(t)$ from the RHS as appropriate, divide by dt and let $dt \to 0$. Balance equations follow by setting derivatives equal to zero.

(1') $$\lambda p_0 = \mu p_1$$

(2') $$(\lambda+\mu)p_1 = \lambda p_0 + 2\mu p_2 + \mu b_2$$

(3') $$(\lambda+2\mu)p_2 = \lambda\bar{\beta}p_1 + 2\mu\bar{\beta}p_3 + \mu\bar{\beta}b_3 \qquad (7.9)$$

(4') $(\lambda+2\mu)p_n = \lambda p_{n-1} + 2\mu\bar{\beta}p_{n+1} + \mu\bar{\beta}b_{n+1}$

(4',a) $2\mu p_J = \lambda p_{J-1}$ for $n = J$

(5') $(\lambda+\mu)b_2 = \lambda\beta p_1 + \mu\beta b_3 + 2\mu\beta p_3$

(6') $(\lambda+\mu)b_n = \lambda b_{n-1} + \mu\beta b_{n+1} + 2\mu\beta p_{n+1}$, $J-1 \geq n \geq 3$

(6',a) $\mu b_J = \lambda b_{J-1}$

Special Case. Let the number of programs in the system be $J = 2$. Then we can solve (1'), (2'), (3'), and (5') simultaneously:

$$p_1 = \frac{\lambda}{\mu} p_0, \quad \frac{\lambda^2}{\mu} p_0 = 2\mu p_2 + \mu b_2, \quad \text{and} \quad \mu b_2 = \lambda\beta p_1$$

yield

$$p_0 = \frac{1}{1 + \frac{\lambda}{\mu} + \left(\frac{\lambda}{\mu}\right)^2\frac{\bar{\beta}}{2} + \left(\frac{\lambda}{\mu}\right)^2\beta}$$

$$b_2 = \left(\frac{\lambda}{\mu}\right)^2 \beta p_0, \quad p_2 = \frac{1}{2}\left(\frac{\lambda}{\mu}\right)^2 \bar{\beta} p_0.$$

The probability of CPU idleness is $b_2 + p_2$, so

$$\text{Expected CPU utilization} = \frac{1 + \frac{\lambda}{\mu}}{1 + \frac{\lambda}{\mu} + \left(\frac{\lambda}{\mu}\right)^2\left[\frac{\bar{\beta}}{2}+\beta\right]} \tag{7.10}$$

Numerical Example. Program material synthesized by B. Wallack of JTSA yielded the values

$$\frac{1}{\mu} = 1.33, \quad \frac{1}{\lambda} = 1.4 \times 1.15 = 1.61$$

$$\frac{\lambda}{\mu} = 0.826$$

Our formula for $J = 2$ shows that if $\beta = \frac{1}{D} = \frac{1}{\#\text{ of disc units}}$,

$$\text{Expected CPU utilization} = \frac{1.83}{1.83 + 0.689\left[\frac{D+1}{2D}\right]},$$

so if

$D = 2$, Expected CPU utilization = 0.78, while if

$D = 4$, Expected CPU utilization = 0.81, and if

$D = \infty$, Expected CPU utilization = 0.84

In order to find CPU idleness probability in the general case, we must find $p_J + b_J$. This can be done by solving equations (1')-(6') simultaneously, subject to the condition that $\sum_{j=0}^{J} p_j + \sum_{j=0}^{J} b_j = 1$. There are in all $(J+1) + (J+1-2) = 2J$ linear equations to be solved; any available package program for linear equation solutions should be applicable.

Another appealing possibility for obtaining a solution is to use an iterative procedure of successive corrections.

C. Approximations

The simplicity of the Markovian channel problem suggests that a good, explicit, approximate solution may be available. One thought is to examine the Disc stage at times when blocking can occur, i.e. when ≥ 2 programs are awaiting Discs (of course, one may be blocked).

Suppose, then, that the Disc stage is "flooded," i.e. that the CPU is fast enough to keep the latter constantly working. Let $P(t)$ denote the probability that the channels are both busy at t, and $B(t)$ denote the probability that one is blocked; clearly $P(t) + B(t) = 1$ when flooding occurs. Now

$$P(t+dt) = P(t)[1-2\mu dt] + P(t)2\mu\bar{\beta}dt + B(t)\mu\bar{\beta}dt \tag{7.11}$$

which leads to

$$\frac{dP}{dt} = -2\mu\ P(t) + 2\mu\bar{\beta}\ P(t) + \mu\bar{\beta}\ B(t)$$

$$= -2\mu\beta\ P(t) + \mu\bar{\beta}[1-P(t)]. \tag{7.12}$$

To find the steady state probability

$$p = \lim_{t\to\infty} P(t)$$

set the derivative equal to zero and solve:

$$p = \frac{\bar{\beta}}{2\beta+\bar{\beta}} = \frac{1-\beta}{1+\beta} = \frac{1 - \frac{1}{D}}{1 + \frac{1}{D}} = \frac{D-1}{D+1} \tag{7.13}$$

and

$$b = \lim_{t\to\infty} B(t) = 1 - p = \frac{2\beta}{1+\beta} = \frac{2}{D+1}\ . \tag{7.14}$$

Consequently the steady state output rate from the "flooded" channels is

$$2\mu p + \mu b = 2\mu\left(\frac{D-1}{D+1}\right) + \mu\,\frac{2}{D+1}$$

$$= 2\mu\,\frac{D}{D+1}\ ,$$

and the effective output rate per channel is

$$\tilde{\mu} = \mu\,\frac{D}{D+1}$$

Now approximate the Disc stage service process as follows:

$$\mu_1 = \mu$$

$$\mu_n = 2\tilde{\mu} \qquad n \geq 2$$

and treat the delays at the Disc stage queue as simple Markovian, with arrivals λ as before and the above service rate. Let d_n denote the steady state probability that n programs are present at Disc Stage ($d_n \approx p_n + b_n$). If this assumption is made,

$$d_n = d_0\,\frac{\lambda_0\,\lambda_1\,\lambda_2\,\cdots\,\lambda_{n-1}}{\mu_1\,\mu_2\,\mu_3\,\cdots\,\mu_n} \tag{7.15}$$

so

$$d_1 = d_0\,\frac{\lambda}{\mu}$$

$$d_2 = d_0\,\frac{\lambda^2}{2\mu\tilde{\mu}}$$

$$d_n = d_0\,\frac{\lambda}{\mu}\left(\frac{\lambda}{2\tilde{\mu}}\right)^{n-1} \qquad n = 1,2,\ldots,J$$

Since

Since

$$1 = \sum_{n=0}^{J} d_n = d_0\left[1 + \frac{\lambda}{\mu}\sum_{n=1}^{J}\left(\frac{\lambda}{2\mu}\right)^{n-1}\right],$$

$$d_0 = \frac{1}{1 + \frac{\lambda}{\mu}\left\{\frac{1-\left(\frac{\lambda}{2\mu}\right)^{J}}{1 - \frac{\lambda}{2\mu}}\right\}}$$

and the probability that all J programs are at the Disc stage is

$$d_J = \frac{\frac{\lambda}{\mu}\left(\frac{\lambda}{2\mu}\right)^{J-1}}{1 + \frac{\lambda}{\mu}\left\{\frac{1-\left(\frac{\lambda}{2\mu}\right)^{J}}{1 - \frac{\lambda}{2\mu}}\right\}} \tag{7.16}$$

then $1 - d_J$ = CPU utilization.

Special Case. Let J = 2. Then

$$d_2 = \frac{\frac{\lambda}{\mu}\cdot\frac{\lambda}{2\mu}}{1 + \frac{\lambda}{\mu}\left(1 + \frac{\lambda}{2\mu}\right)}$$

and

$$1 - d_2 = \frac{1 + \frac{\lambda}{\mu}}{1 + \frac{\lambda}{\mu} + \frac{\lambda}{\mu}\frac{\lambda}{2\mu}}$$

$$= \frac{1 + \frac{\lambda}{\mu}}{1 + \frac{\lambda}{\mu} + \frac{1}{2}\left(\frac{\lambda}{\mu}\right)^2\left(\frac{D+1}{D}\right)}$$

which agrees with the answer found by using the correct Markovian analysis for this case. It thus seems reasonable to use the approximate result d_J in order to estimate CPU utilization.

REFERENCES

[1] Buzen, J., "Computational algorithms for closed queueing networks with exponential servers," Comm. of ACM, Vol. 16, Sept. 1973.

[2] Cox, D. R. and Smith, W., Queues, John Wiley (Methuen Monograph) 1961.

[3] Gaver, D. P., "Probability models for multiprogramming computer systems," J. Assoc. Computing Mach., Vol. 14, 1967.

[4] Gaver, D. P. and Thompson, G. L., Programming and Probability Models in Operations Research, Brooks-Cole Pub. Co., Monterey, Calif., 1973.

[5] Gaver, D. P. and Shedler, G., "Approximate models for processor utilization in multiprogrammed computer systems," SIAM J. Computing, Vol. 2, No. 3, Sept. 1973.

[6] Gordon, W. and Newell, G., "Closed queueing systems with exponential servers," Opns. Research, Vol. 15, 1967.

[7] Lewis, P. A. W. and Shedler, G., "A cyclic-queue model of system overhead in multiprogrammed computer systems," J. Assoc. Computing Mach., Vol. 18, 1971.

[8] Ross, S., Applied Probability Models with Optimization Application, Holden-Day, 1970.

INITIAL DISTRIBUTION LIST

	Copies
Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
Dean of Research Code 023 Naval Postgraduate School Monterey, California 93940	1
Library (Code 0212) Naval Postgraduate School Monterey, California 93940	2
Library (Code 55) Naval Postgraduate School Monterey, California 93940	2
C. Giammo Defense Communication Agency JTSA 11440 Isaac Newton Square Reston, Virginia 22090	5
B. Wallach Defense Communication Agency JTSA 11440 Isaac Newton Square Reston, Virginia 22090	2
P. Kiviat FEDSIM Washington, D. C. 20330	3
M. Spiegel FEDSIM Washington, D. C. 20330	2
L. Kleinrock Computer Science Department UCLA Los Angeles, California 90024	1
R. Muntz Computer Science Department UCLA Los Angeles, California 90024	1

E. Gelenbe 1
IRIA (Laboria)
Domaine de Voluceau - Rocquecourt
B.P. 5 - 78150 Le Chesnay
France

G. Shedler 1
IBM Research
San Jose, California 95100

W. Tuel 1
IBM Research
San Jose, California 95100

K. Marshall 1
P. Milch 1
M. Thomas 1
R. Butterworth 1
D. Gaver 15
Code 55
Naval Postgraduate School
Monterey, California 93940

Technical Library 1
Naval Ordnance Station
Indian Head, Maryland 20640

Mr. W. L. Nicholson 1
Staff Scientist
Pacific Northwest Laboratories
Batelle Boulevard
Richland, Washington 99352

Dr. W. James White 1
Info Results, Ltd.
Suite 204, 2074 Lawrence Avenue West
Toronto, Ontario
Canada

Dr. Jackson L. Albuquerque
Dept. de Estatística Cx. Postal 1103
Universidade Federal do Ceará
60.000 Fortaleza, Ceará
Brasil